Binomial Naive bayes

In [2]:

```
#import pickle

#favorate_sample = ['appple','mango','banana','mangoose']

#pickle.dump( favorate_sample, open('sal.p', 'wb'))
#favorate_sample = pickle.load(open('sal.p', 'rb'))

#favorate_sample
```

Out[2]:

```
['appple', 'mango', 'banana', 'mangoose']
```

In [4]:

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import pickle

# importing filtered data with BOW from previous exercise

bow_data = pickle.load(open('BOW.p','rb')) ##
print(bow_data.shape)
print(type(bow_data))

cleaned_data = pickle.load(open('clen.p','rb')) # importing data with time split

cleaned_data.head(5)
```

```
(28505, 25785)
<class 'scipy.sparse.csr.csr_matrix'>
```

Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	positive	9
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	positive	9
417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	positive	9
346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	2	positive	9
417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0	positive	9

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score

In [5]:

```
def partition(x):
    if x == 'negative':
        return '0'
    return '1'
actualScore = cleaned_data['Score']
positiveNegative = actualScore.map(partition)
cleaned_data['Score'] = positiveNegative
```

In [6]:

```
positiveNegative = cleaned_data['Score']
print (positiveNegative.value_counts())
positiveNegative.shape

cleaned_data.head(5)
```

```
1    212786
0     37214
Name: Score, dtype: int64
```

Out[6]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	1	93
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	1	94
417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	1	94
346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	2	1	94
417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0	1	94

In [7]:

```
positive_negative= positiveNegative.map(lambda x: 1 if int (x) is 1 else 0)
print (positive_negative.value_counts())
```

```
1    212786
0     37214
Name: Score, dtype: int64
```

In [8]:

```
positive_negative.value_counts()
```

Out[8]:

```
1    212786
0     37214
Name: Score, dtype: int64
```

In [9]:

```
# Importing sklearn libraries for BOW
from sklearn.preprocessing import StandardScaler
from sklearn.cross_validation import train_test_split
#from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import cross_val_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score
from sklearn import cross_validation

process = cleaned_data['CleanedText']
```

In [10]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
```

In [11]:

```
X_tra, X_tes, y_train, y_test = train_test_split(cleaned_data['CleanedText'].values,positiveNegativ
e.values,test_size=0.3,shuffle=False)

#Implementing BAG of words
tf_idf_vect = TfidfVectorizer(ngram_range=(1,0),dtype=float)
X_BOW = tf_idf_vect.fit_transform(X_tra)
# Standerdising the data
norm = StandardScaler(with_mean = False)
X_train = norm.fit_transform(X_BOW)
#X_train = preprocessing.normalize(X_BOW)

#vectorizer for test data
X_BOW1 = tf_idf_vect.transform(X_tes)
# Standerdising the data
X_test = norm.transform(X_BOW1)

from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)
for train, cv in tscv.split(X_train):
    print(X_train[train].shape, X_train[cv].shape)

y_train.shape,X_train.shape
```

```
(15910, 73789) (15909, 73789)
(31819, 73789) (15909, 73789)
(47728, 73789) (15909, 73789)
(63637, 73789) (15909, 73789)
(79546, 73789) (15909, 73789)
(95455, 73789) (15909, 73789)
(111364, 73789) (15909, 73789)
(127273, 73789) (15909, 73789)
(143182, 73789) (15909, 73789)
(159091, 73789) (15909, 73789)
```

Out[11]:

```
((175000,), (175000, 73789))
```

In [13]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import BernoulliNB
```

```
from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer

tscv = TimeSeriesSplit(n_splits=3)
bnb = BernoulliNB()
f1 = make_scorer(f1_score, pos_label='1')
param_grid = {'alpha':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]}
gsv = GridSearchCV(bnb,param_grid,cv=tscv,verbose=1,scoring=f1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best f1: %.2f%%"%(gsv.best_score_*100))


#gsv.accuracy_score()
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
```

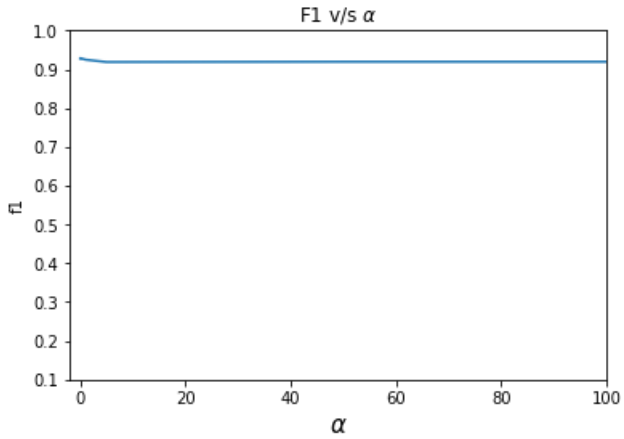
```
[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed:  1.1min finished
```

```
Best HyperParameter:  {'alpha': 0.005}
Best f1: 92.88%
```

In [15]:

```
import matplotlib.pyplot as plt
x=[]
y=[]
for a in gsv.grid_scores_:
    x.append(a[0]['alpha'])
    y.append(a[1])
plt.xlim(-2,100)
plt.ylim(0.1,1)
plt.xlabel(r"$\alpha$",fontsize=15)
plt.ylabel("f1")
plt.title(r'F1 v/s $\alpha$')
plt.plot(x,y)
plt.show()
```

```
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in version 0.18 in favor of the more
elaborate cv_results_ attribute. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```

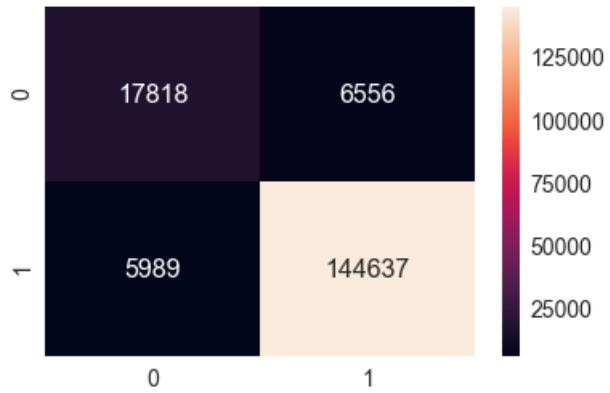


In [16]:

```
# train data
from sklearn.metrics import make_scorer
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
import seaborn as sns

# Train data
bnb = BernoulliNB(alpha=0.005)
```

```
bnb.fit(X_train,y_train)
y_pre = bnb.predict(X_train)

print (classification_report(y_train,y_pre))
result = confusion_matrix(y_train,y_pre)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)

sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

print("f1 on test set: %0.3f%%"%(f1_score(y_train, y_pre, pos_label='1')*100))
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_train, y_pre)*100))
```

```
             precision    recall  f1-score   support

          0       0.75      0.73      0.74     24374
          1       0.96      0.96      0.96    150626

avg / total       0.93      0.93      0.93    175000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 17818   6556]
 [  5989 144637]]
f1 on test set: 95.844%
Accuracy on test set: 92.831%
```



In [60]:

```
# Find the train error############
accuracy_score(y_train, y_pre)
B_train_error = [1 - 0.9267142857142857]
B_train_error
```

Out[60]:

```
[0.07328571428571429]
```

In [18]:

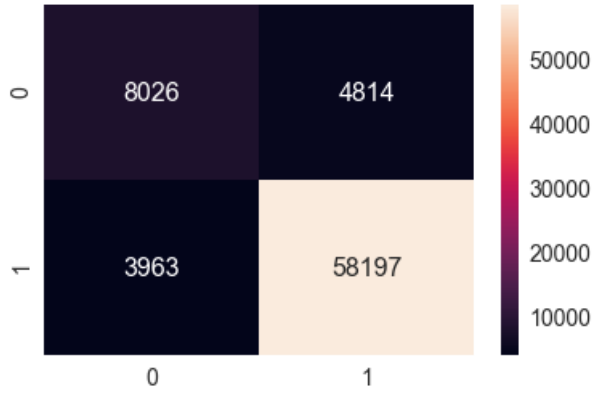
```
# Test data

bnb.fit(X_train,y_train)
y_pred = bnb.predict(X_test)
#y_predict =bnb.(X_test)

print (classification_report(y_test,y_pred))
result = confusion_matrix(y_test,y_pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

print("f1 on test set: %0.3f%%"%(f1_score(y_test, y_pred,pos_label='1')*100))
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
```

```
# Find the test error
accuracy_score(y_test, y_pred)
B_test_error = [1 - 0.8897466666666667]
B_test_error
```

```
             precision    recall  f1-score   support

          0       0.67      0.63      0.65     12840
          1       0.92      0.94      0.93     62160

avg / total       0.88      0.88      0.88     75000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 8026  4814]
 [ 3963 58197]]
f1 on test set: 92.988%
Accuracy on test set: 88.297%
```

Out[18]:

```
[0.11025333333333331]
```



In [64]:

```
import pandas as pd

#getting all feature names
all_feat = tf_idf_vect.get_feature_names()
#getting feature_log_prob of bernoulli naive bayes
log_probabilities=bnb.feature_log_prob_

neg_imp = pd.DataFrame({'words':all_feat,'Negative_probability':log_probabilities[0]})
neg_imp.head(5)
```

Out[64]:

	words	Negative_probability
0	aa	-15.399590
1	aaa	-10.096285
2	aaaa	-15.399590
3	aaaaaaaaaaaaaaaaaaargh	-15.399590
4	aaaaaaaaagghh	-15.399590

In [65]:

```
#sorting in descending order
neg_imp = neg_imp.sort_values(by="Negative_probability",ascending=False)
neg_imp.head(5)
```

Out[65]:

	words	Negative_probability
63411	tast	-0.994073
36721	like	-1.021724
50470	product	-1.209210
44880	one	-1.352015
72429	would	-1.451298

In [66]:

```
#dataframe for positive probabilites
pos_imp = pd.DataFrame({"word":all_feat,"positive_probability":log_probabilities[1]})
pos_imp.head(5)
```

Out[66]:

	word	positive_probability
0	aa	-11.226911
1	aaa	-10.822278
2	aaaa	-11.917568
3	aaaaaaaaaaaaaaaaaaargh	-11.917568
4	aaaaaaaaagghh	-11.917568

In [67]:

```
#sorting in descending order
pos_imp = pos_imp.sort_values(by="positive_probability",ascending=False)
pos_imp.head(5)
```

Out[67]:

	word	positive_probability
36721	like	-1.204234
63411	tast	-1.216685
27336	good	-1.292075
37572	love	-1.292703
28029	great	-1.310933

Multimonial naive bayes

In [74]:

```
X_tra, X_tes, y_train, y_test = train_test_split(cleaned_data['CleanedText'].values,positive_negati
ve,test_size=0.3,shuffle=False)

#Implementing BAG of words
bi_gram = TfidfVectorizer(ngram_range=(1,2))
X_BOW = bi_gram.fit_transform(X_tra)
# Standerdising the data
norm = StandardScaler(with_mean = False)
X_train = norm.fit_transform(X_BOW)
#X_train = preprocessing.normalize(X_BOW)

#vectorizer for test data
X_BOW1 = bi_gram.transform(X_tes)
# Standerdising the data
X_test = norm.transform(X_BOW1)
```

```
from sklearn.model_selection import TimeSeriesSplit

for train, cv in tscv.split(X_train):
    print(X_train[train].shape, X_train[cv].shape)
```

```
(35000, 1854573) (35000, 1854573)
(70000, 1854573) (35000, 1854573)
(105000, 1854573) (35000, 1854573)
(140000, 1854573) (35000, 1854573)
```

In [77]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import RandomizedSearchCV

tscv = TimeSeriesSplit(n_splits=4) #For time based splitting
mnb = MultinomialNB()

param_grid = {'alpha':[10,1,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001,0.00005]}
rsv = RandomizedSearchCV(mnb,param_grid,cv=tscv,verbose=1,scoring='f1')
rsv.fit(X_train,y_train)
print("Best HyperParameter: ",rsv.best_params_)
print("Best f1: %.2f%%"%(rsv.best_score_*100))
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
```

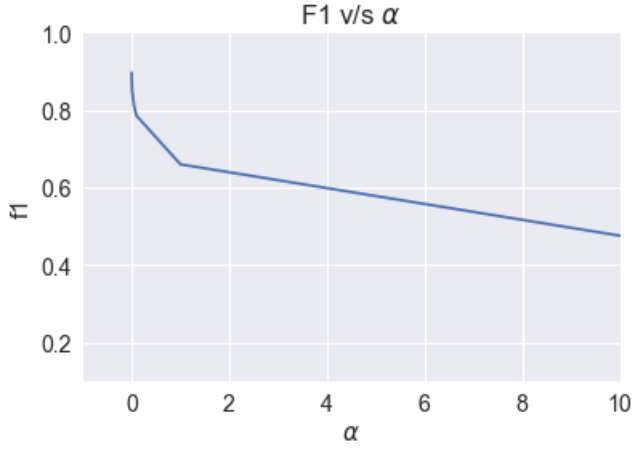
```
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:   45.3s finished
```

```
Best HyperParameter:  {'alpha': 5e-05}
Best f1: 89.69%
```

In [80]:

```
import matplotlib.pyplot as plt
x=[]
y=[]
for a in rsv.grid_scores_:
    x.append(a[0]['alpha'])
    y.append(a[1])
plt.xlim(-1,10)
plt.ylim(0.1,1)
plt.xlabel(r"$\alpha$",fontsize=15)
plt.ylabel("f1")
plt.title(r'F1 v/s $\alpha$')
plt.plot(x,y)
plt.show()
```

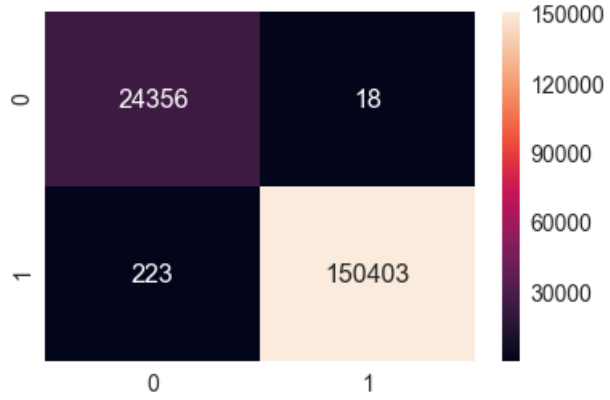
```
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in version 0.18 in favor of the more
elaborate cv_results_ attribute. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```



In [81]:

```
# train data
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
import seaborn as sns

# Train data
mnb = MultinomialNB(alpha=0.00005)
mnb.fit(X_train,y_train)
y_pre = mnb.predict(X_train)

print (classification_report(y_train,y_pre))
result = confusion_matrix(y_train,y_pre)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)

sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

print("f1 on test set: %0.3f%%"%(f1_score(y_train, y_pre)*100))
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_train, y_pre)*100))
```

```
             precision    recall  f1-score   support

          0       0.99      1.00      1.00     24374
          1       1.00      1.00      1.00    150626

avg / total       1.00      1.00      1.00    175000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 24356     18]
 [   223 150403]]
f1 on test set: 99.920%
Accuracy on test set: 99.862%
```



In [83]:

```
# Find the train error############
accuracy_score(y_train, y_pre)
B_train_error = [1 - 0.9986228571428571]
B_train_error
```

Out[83]:

```
[0.0013771428571428546]
```

In [89]:

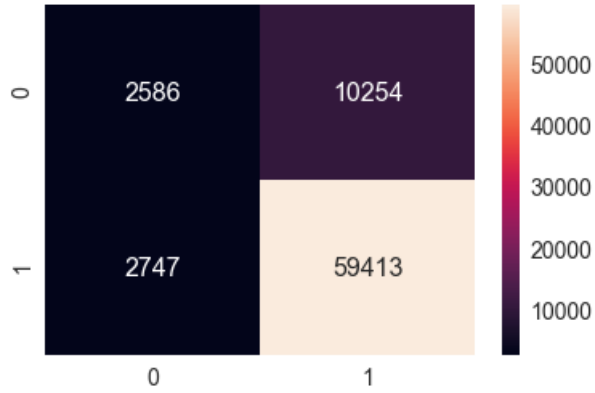
```
# Test data

mnb.fit(X_train,y_train)
y_pred = mnb.predict(X_test)
#y_predict =bnb.(X_test)
```

```
print (classification_report(y_test,y_pred))
result = confusion_matrix(y_test,y_pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

print("f1 on test set: %0.3f%%"%(f1_score(y_test, y_pred)*100))
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))

# Find the test error
accuracy_score(y_test, y_pred)
B_test_error = [1 - 0.8266533333333334]
B_test_error
```

```
             precision    recall  f1-score   support

          0       0.48      0.20      0.28     12840
          1       0.85      0.96      0.90     62160

avg / total       0.79      0.83      0.80     75000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 2586 10254]
 [ 2747 59413]]
f1 on test set: 90.138%
Accuracy on test set: 82.665%
```

Out[89]:

```
[0.17334666666666665]
```



In [19]:

```
# Binnary BOW
#Implementing BAG of words
bi_gram = CountVectorizer(1,0)
X_BOW = bi_gram.fit_transform(X_tra)
# Standerdising the data
norm = StandardScaler(with_mean = False)
X_train = norm.fit_transform(X_BOW)
#X_train = preprocessing.normalize(X_BOW)

#vectorizer for test data
X_BOW1 = bi_gram.transform(X_tes)
# Standerdising the data
X_test = norm.transform(X_BOW1)

from sklearn.model_selection import TimeSeriesSplit

for train, cv in tscv.split(X_train):
    print(X_train[train].shape, X_train[cv].shape)
```

```
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
  warnings.warn(msg, DataConversionWarning)
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

```
(43750, 73789) (43750, 73789)
(87500, 73789) (43750, 73789)
(131250, 73789) (43750, 73789)
```

In [20]:

```
tscv = TimeSeriesSplit(n_splits=3)
bnb = BernoulliNB()
f1 = make_scorer(f1_score, pos_label='1')
param_grid = {'alpha':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]}
gsv = GridSearchCV(bnb,param_grid,cv=tscv,verbose=1,scoring=f1)
gsv.fit(X_train,y_train)
print("Best HyperParameter: ",gsv.best_params_)
print("Best f1: %.2f%%"%(gsv.best_score_*100))
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
```

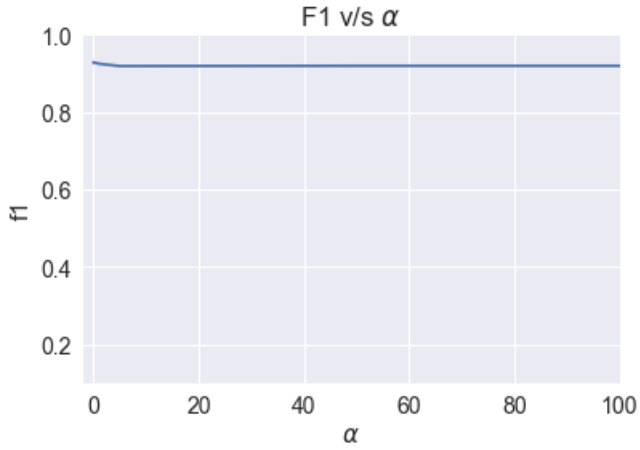
```
[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed:  1.1min finished
```

```
Best HyperParameter:  {'alpha': 0.005}
Best f1: 92.88%
```

In [21]:

```
import matplotlib.pyplot as plt
x=[]
y=[]
for a in gsv.grid_scores_:
    x.append(a[0]['alpha'])
    y.append(a[1])
plt.xlim(-2,100)
plt.ylim(0.1,1)
plt.xlabel(r"$\alpha$",fontsize=15)
plt.ylabel("f1")
plt.title(r'F1 v/s $\alpha$')
plt.plot(x,y)
plt.show()
```

```
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in version 0.18 in favor of the more
elaborate cv_results_ attribute. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```

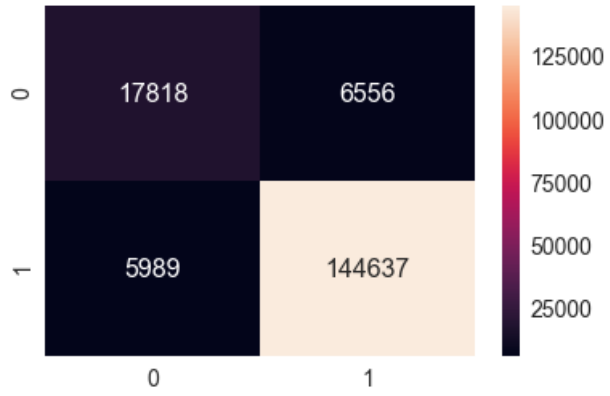


In [22]:

```
# Train data
```

```
bnb = BernoulliNB(alpha=0.005)
bnb.fit(X_train,y_train)
y_pre = bnb.predict(X_train)

print (classification_report(y_train,y_pre))
result = confusion_matrix(y_train,y_pre)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)

sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

print("f1 on test set: %0.3f%%"%(f1_score(y_train, y_pre, pos_label='1')*100))
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_train, y_pre)*100))
```

```
             precision    recall  f1-score   support

          0       0.75      0.73      0.74     24374
          1       0.96      0.96      0.96    150626

avg / total       0.93      0.93      0.93    175000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 17818   6556]
 [  5989 144637]]
f1 on test set: 95.844%
Accuracy on test set: 92.831%
```

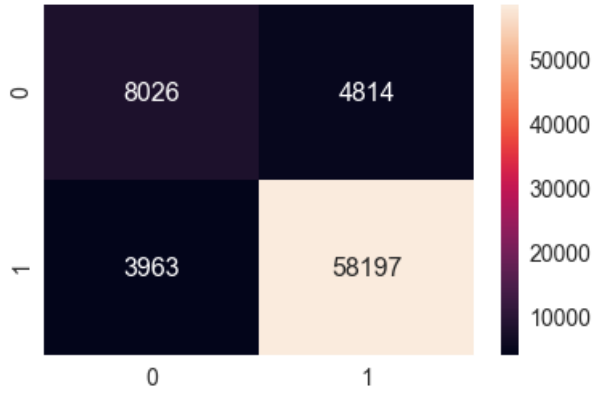


In [23]:

```
bnb.fit(X_train,y_train)
y_pred = bnb.predict(X_test)
#y_predict =bnb.(X_test)

print (classification_report(y_test,y_pred))
result = confusion_matrix(y_test,y_pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

print("f1 on test set: %0.3f%%"%(f1_score(y_test, y_pred,pos_label='1')*100))
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
```

```
             precision    recall  f1-score   support

          0       0.67      0.63      0.65     12840
          1       0.92      0.94      0.93     62160

avg / total       0.88      0.88      0.88     75000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 8026  4814]
```

```
 [ 3963 58197]]
f1 on test set: 92.988%
Accuracy on test set: 88.297%
```



Bernoulli Naive bayes(TFIDF)	**Multimonial Naive bayes**	**Bernoulli Naive bayes(BOW)**
1. Best Hyperparameter of Alpha is @ 0.005	Best Hyperparameter of Alpha is @ 0.00005	Best Hyperparameter of Alpha is @ 0.005
2. F1 score of Train data	F1 score of Train data	F1 score of Train data
95.74%	93.68%	95.84%
3. F1 score of Test data	F1 score of Test data	F1 score of Test data
93.68%	90.13%	92.98%